Linear Regression II, SGD

Milan Straka

October 11, 2021

Charles University in Prague
Faculty of Mathematics and Physics
Institute of Formal and Applied Linguistics

unless otherwise stated

Linear Regression

Given an input value $\boldsymbol{x} \in \mathbb{R}^D$, **linear regression** computes predictions as:

$$y(\boldsymbol{x}; \boldsymbol{w}, b) = x_1 w_1 + x_2 w_2 + \ldots + x_D w_D + b = \sum_{i=1}^{D} x_i w_i + b = \boldsymbol{x}^T \boldsymbol{w} + b.$$

The *bias* b can be considered one of the *weights* $\boldsymbol{w}$ if convenient.

We train the weights by minimizing an **error function** between the real target values and their predictions, notably *sum of squares*:

$$\frac{1}{2} \sum_{i=1}^{N} \big(y(\boldsymbol{x}_i; \boldsymbol{w}) - t_i\big)^2$$

There are several ways how to minimize it, but in our case, there exists an explicit solution:

$$\boldsymbol{w} = (\boldsymbol{X}^T \boldsymbol{X})^{-1} \boldsymbol{X}^T \boldsymbol{t}.$$

Linear Regression Example

Assume our input vectors comprise of $\boldsymbol{x} = (x^0, x^1, \ldots, x^M)$, for $M \geq 0$.

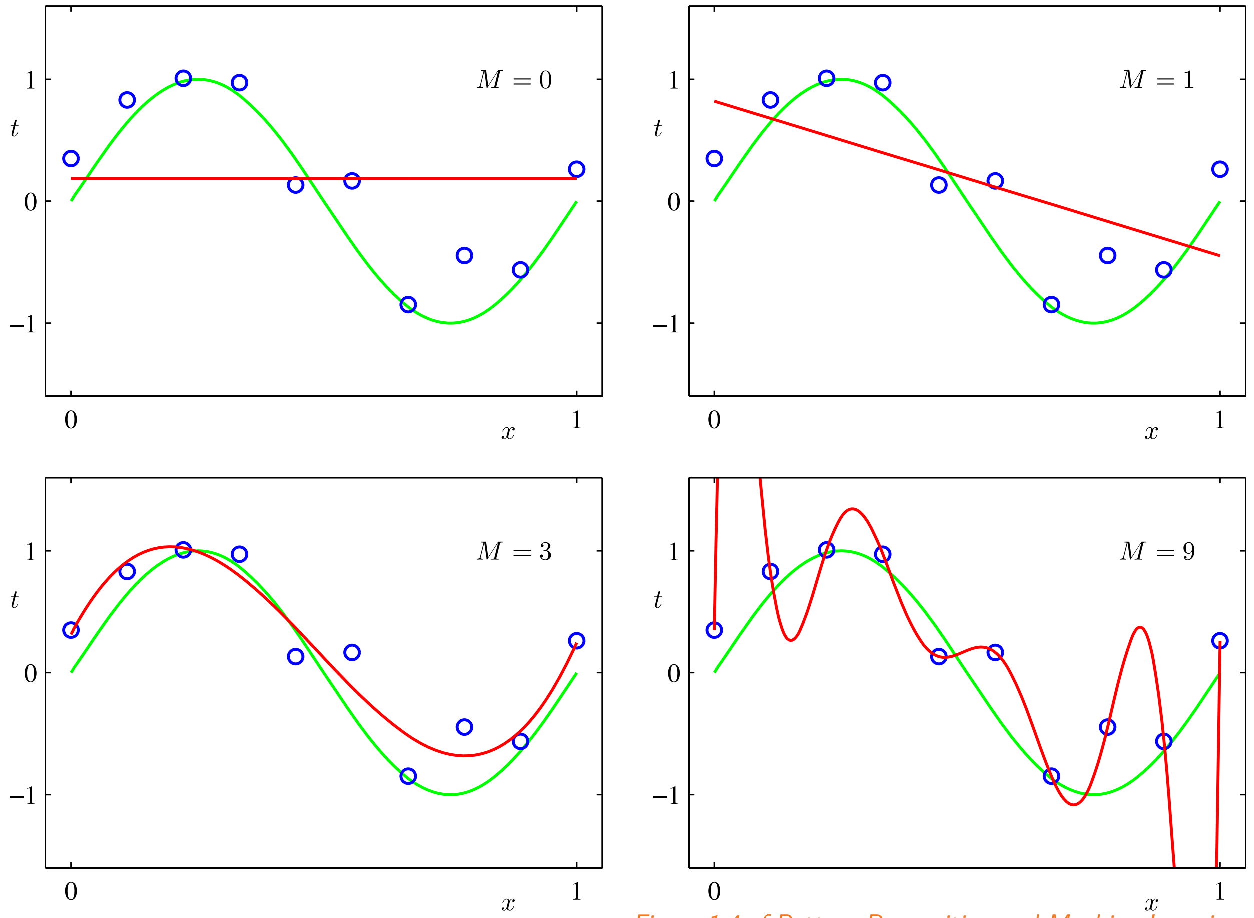


Figure 1.4 of Pattern Recognition and Machine Learning.

Linear Regression Example

To plot the error, the *root mean squared error* $\mathrm{RMSE} = \sqrt{\mathrm{MSE}}$ is frequently used.

The displayed error nicely illustrates two main challenges in machine learning:

- *underfitting*
- *overfitting*

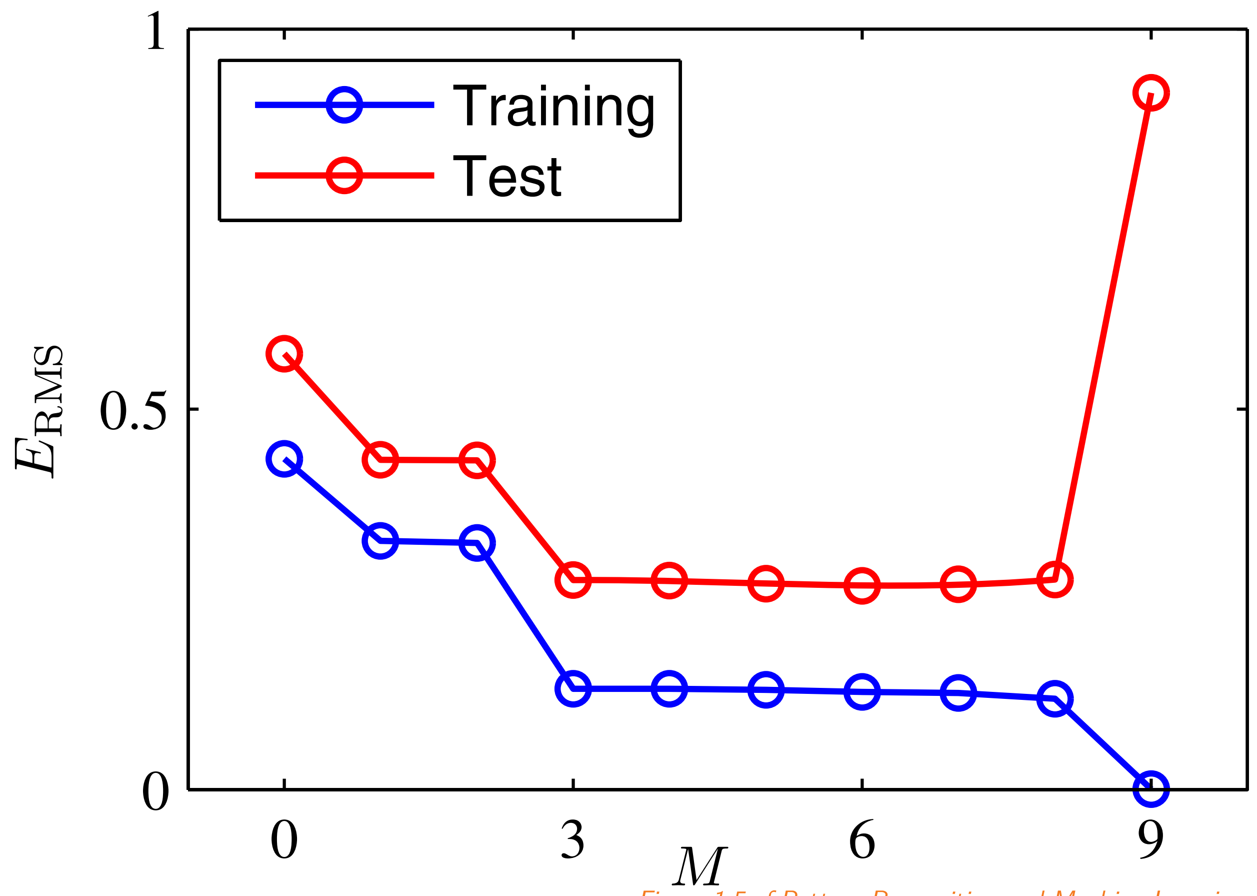


Figure 1.5 of Pattern Recognition and Machine Learning.

Model Capacity

We can control whether a model underfits or overfits by modifying its *capacity*.

- representational capacity
- effective capacity

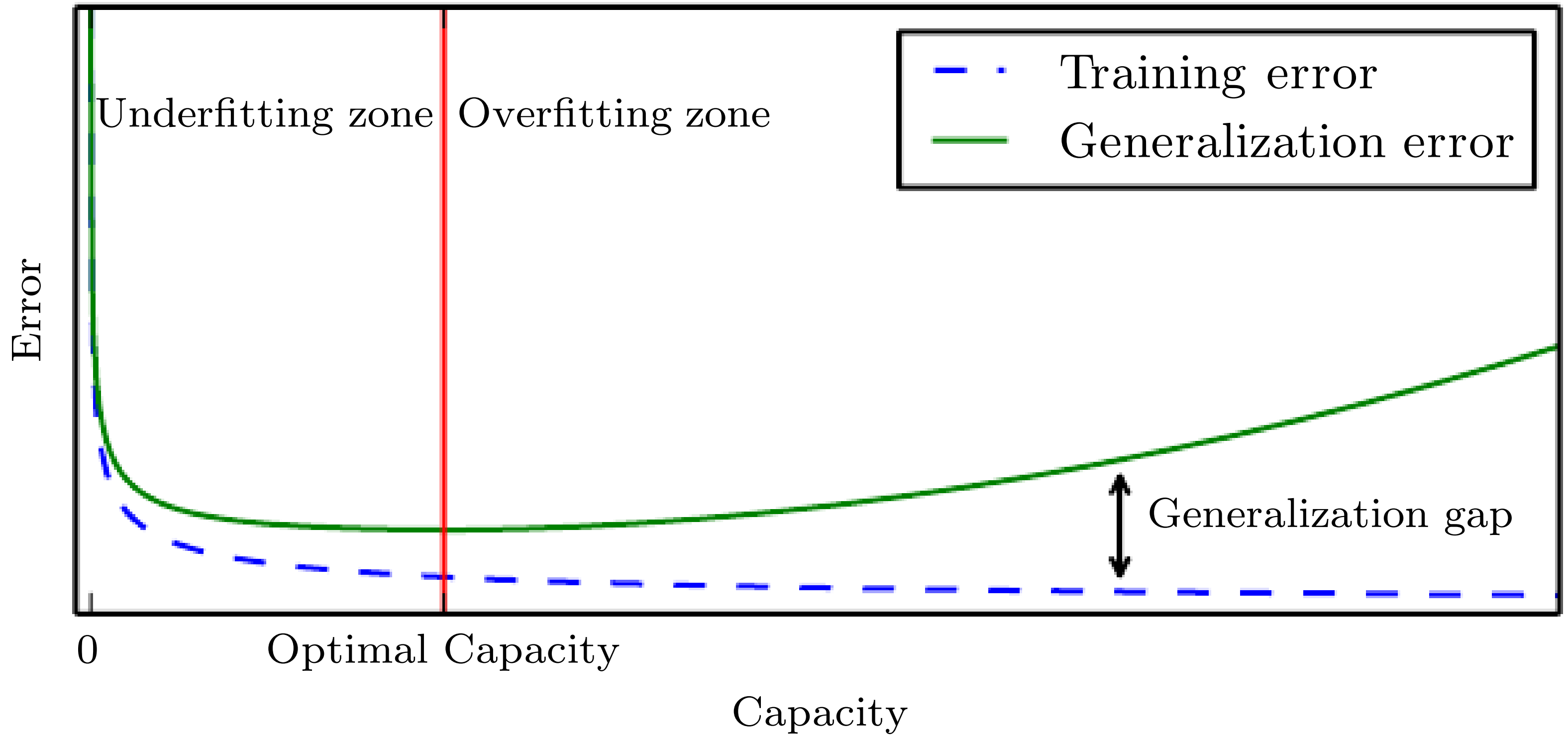


Figure 5.3, page 115 of Deep Learning Book, http://deeplearningbook.org

Linear Regression Overfitting

Note that employing more data usually alleviates overfitting (the relative capacity of the model is decreased).

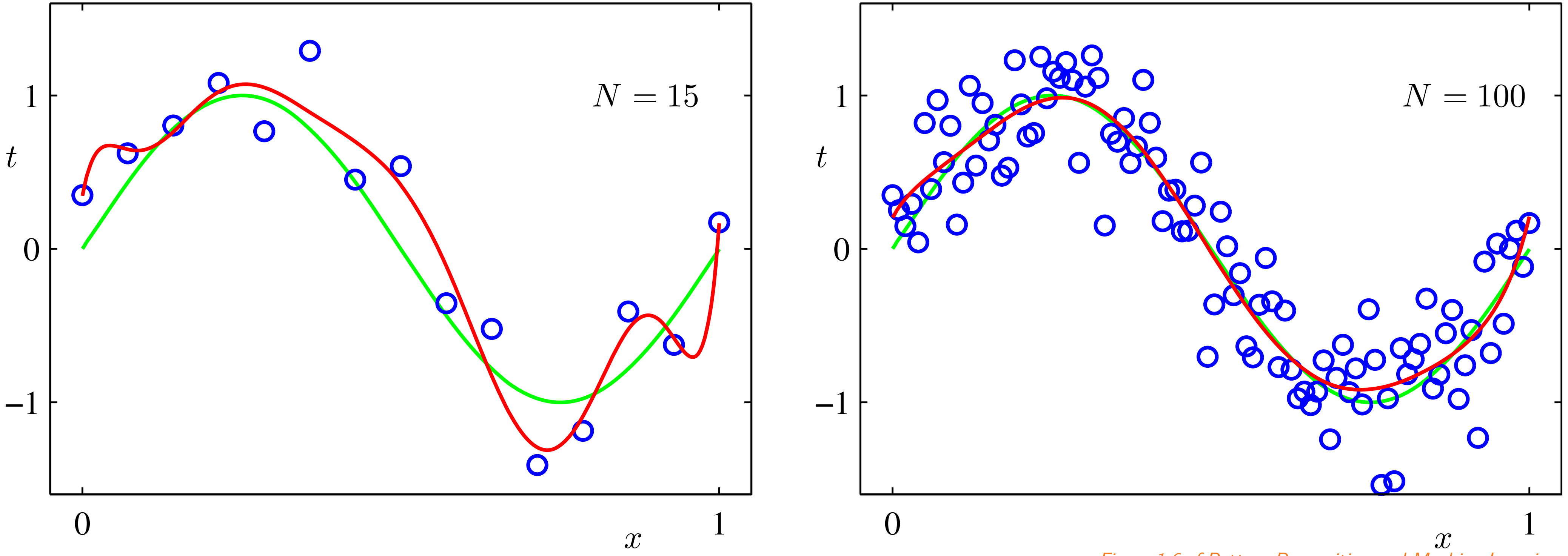
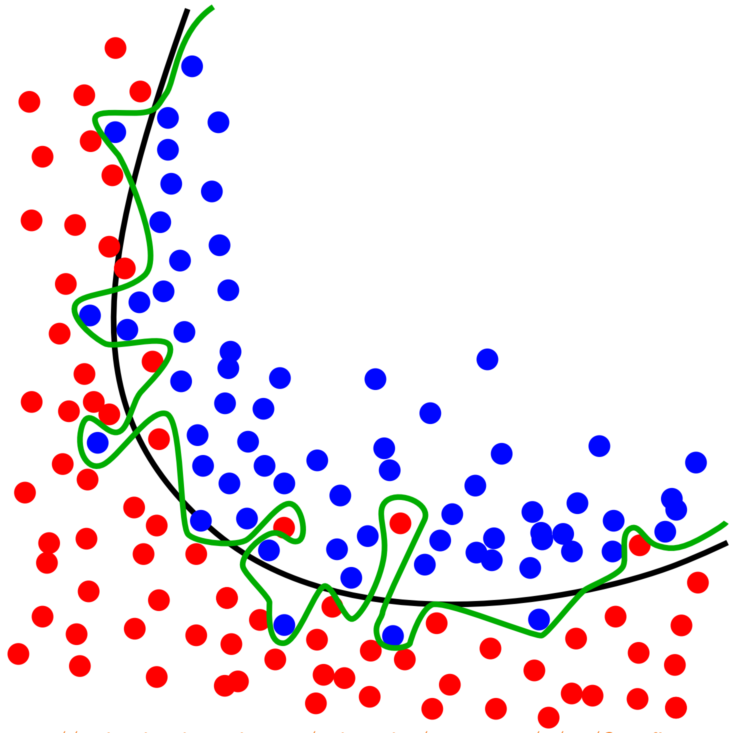


Figure 1.6 of Pattern Recognition and Machine Learning.

Regularization

Regularization in a broad sense is any change in a machine learning algorithm that is designed to *reduce generalization error* (but not necessarily its training error).

We already saw that **limiting model capacity** can work as regularization.



https://upload.wikimedia.org/wikipedia/commons/1/19/Overfitting.svg

L2 Regularization

One of the oldest regularization techniques tries to prefer “simpler” models by endorsing models with **smaller weights**.

Concretely, $\boldsymbol{L_2}$**-regularization** (also called **weight decay**) penalizes models with large weights by utilizing the following error function:

$$\frac{1}{2}\sum_{i=1}^{N}\big(y(\boldsymbol{x}_i;\boldsymbol{w}) - t_i\big)^2 + \frac{\lambda}{2}\|\boldsymbol{w}\|^2$$

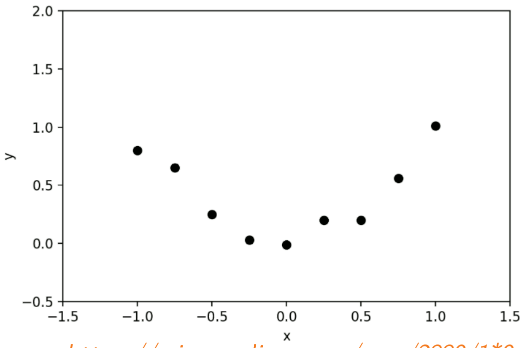
Note that the L_2-regularization usually is not applied on *bias*, only on the “proper” weights.
One of the reasons for this is that without penalizing bias, L_2-regularization is invariant to shifts (i.e., adding a constant to all targets would result in the same solution with only bias increased by that constant; if bias would be penalized, this would not be true).

For simplicity, we will not explicitly exclude the bias from the L_2-regularization penalty in the slides (several textbooks also take the same approach).

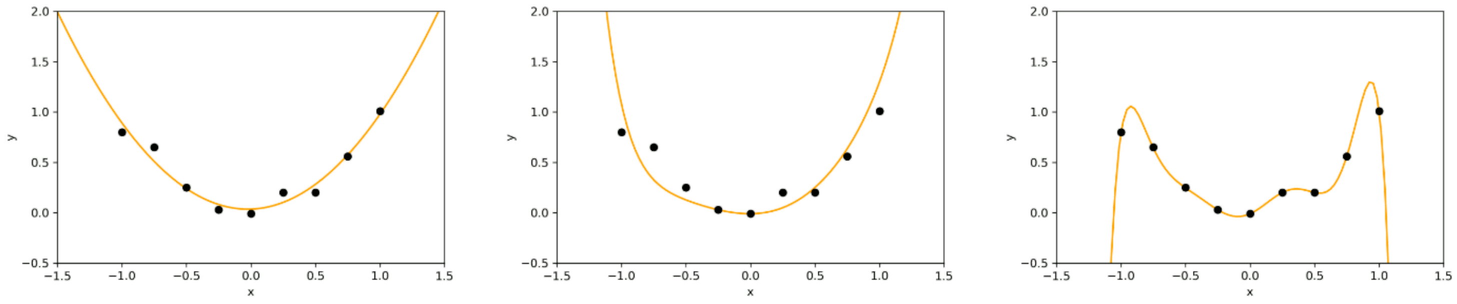
L2 Regularization

One way how you can look at L_2-regularization is that it promotes smaller changes of the model (the gradient of linear regression with respect to the inputs are exactly the weights, i.e., $\nabla_{\boldsymbol{x}} y(\boldsymbol{x}; \boldsymbol{w}) = \boldsymbol{w}$).

Considering the data points on the right, we present mean squared errors and L_2 norms of the weights for three linear regression models:



https://miro.medium.com/max/2880/1*0-fsK9RkqL3rogo2SnZPCg.png



(a) #params = 3
MSE = 0.006
L2 norm = 0.90
L1 norm = 0.98

(b) #params = 9
MSE = 0.035
L2 norm = 1.06
L1 norm = 2.32

(c) #params = 9
MSE = 0
L2 norm = 32.69
L1 norm = 70.03

https://miro.medium.com/max/2880/1*DVFYChNDMNIS_7CVq2PhSQ.png

Figure a: $\hat{y} = 0.04 + 0.04x + 0.9x^2$
Figure b: $\hat{y} = -0.01 + 0.01x + 0.8x^2 + 0.5x^3 - 0.1x^4 - 0.1x^5 + 0.3x^6 - 0.3x^7 + 0.2x^8$
Figure c: $\hat{y} = -0.01 + 0.57x + 2.67x^2 - 4.08x^3 - 12.25x^4 + 7.41x^5 + 24.87x^6 - 3.79x^7 - 14.38x^8$

https://miro.medium.com/max/2880/1*UolRIKXikCz7SFsPfSZrYQ.png

L2 Regularization

The effect of L_2-regularization can be seen as limiting the *effective capacity* of the model.

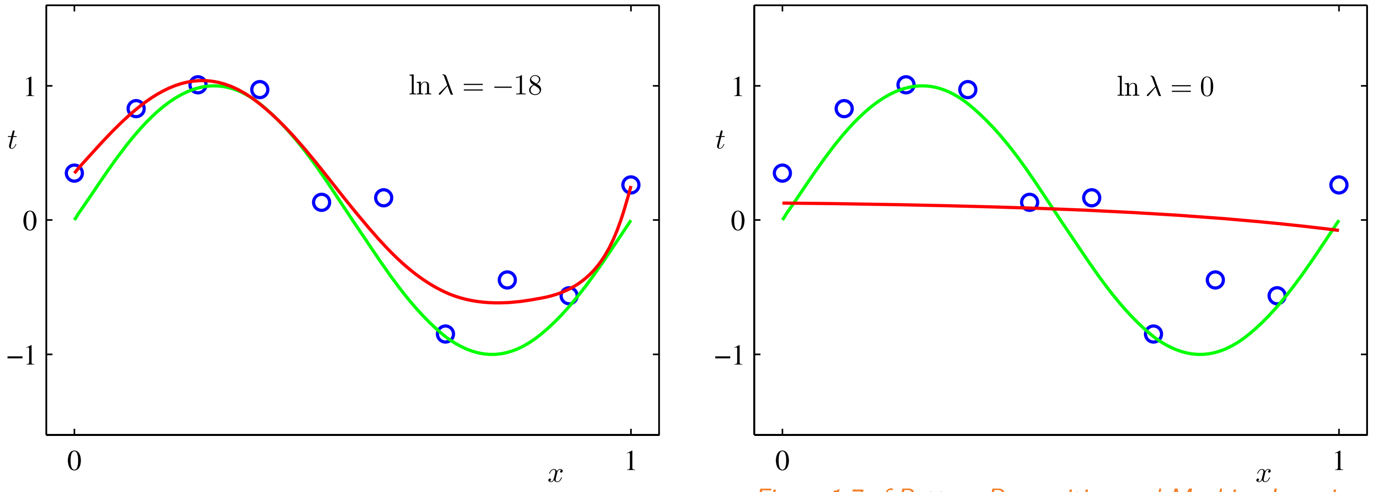


Figure 1.7 of Pattern Recognition and Machine Learning.

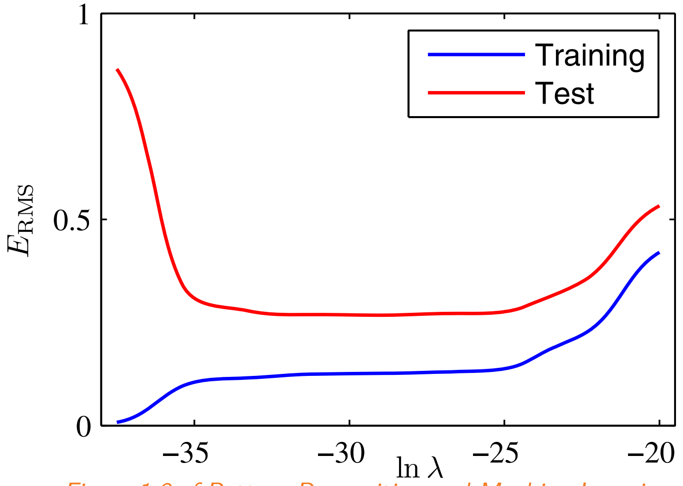


Figure 1.8 of Pattern Recognition and Machine Learning.

Regularizing Linear Regression

In matrix form, regularized sum of squares error for linear regression amounts to

$$\frac{1}{2}\|\boldsymbol{X}\boldsymbol{w}-\boldsymbol{t}\|^2+\frac{\lambda}{2}\|\boldsymbol{w}\|^2.$$

When repeating the same calculation as in the unregularized case, we arrive at

$$(\boldsymbol{X}^T\boldsymbol{X}+\lambda\boldsymbol{I})\boldsymbol{w}=\boldsymbol{X}^T\boldsymbol{t},$$

where $\boldsymbol{I}$ is an identity matrix.

Input: Dataset ($\boldsymbol{X}\in\mathbb{R}^{N\times D}$, $\boldsymbol{t}\in\mathbb{R}^N$), constant $\lambda\in\mathbb{R}^+$.
Output: Weights $\boldsymbol{w}\in\mathbb{R}^D$ minimizing MSE of regularized linear regression.

- $\boldsymbol{w}\leftarrow(\boldsymbol{X}^T\boldsymbol{X}+\lambda\boldsymbol{I})^{-1}\boldsymbol{X}^T\boldsymbol{t}.$

Note that the $\boldsymbol{X}^T\boldsymbol{X}+\lambda\boldsymbol{I}$ matrix is always regular for $\lambda>0$, so another effect of L_2-regularization is that the inverse always exists.

Choosing Hyperparameters

Hyperparameters are not adapted by the learning algorithm itself.

Usually a **validation set** or **development set** is used to estimate the generalization error, allowing to update hyperparameters accordingly. If there is not enough data (well, there is **always** not enough data), more sophisticated approaches can be used.

So far, we have seen two hyperparameters, M and λ.

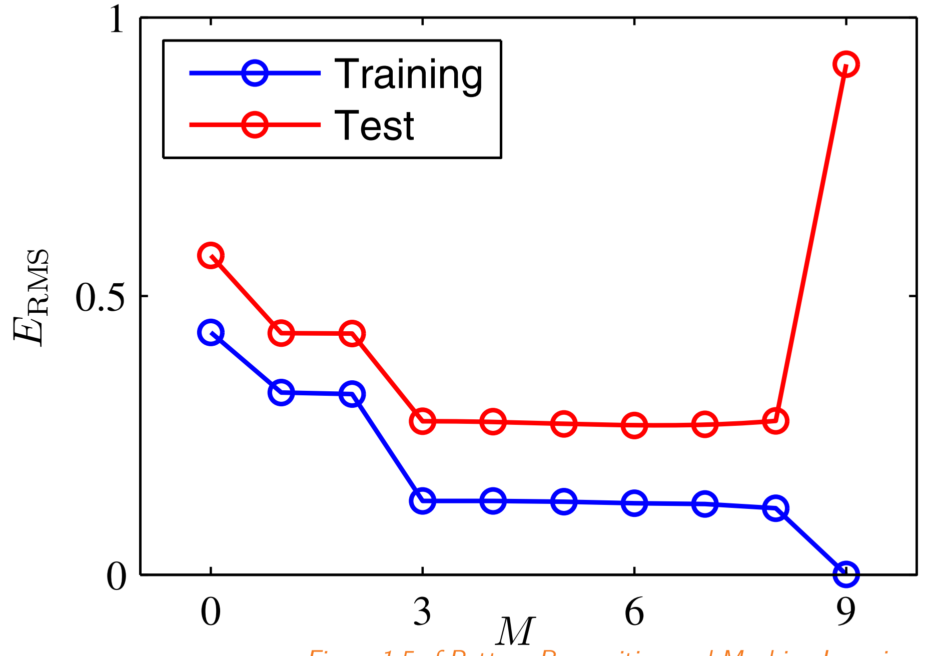


Figure 1.5 of Pattern Recognition and Machine Learning.

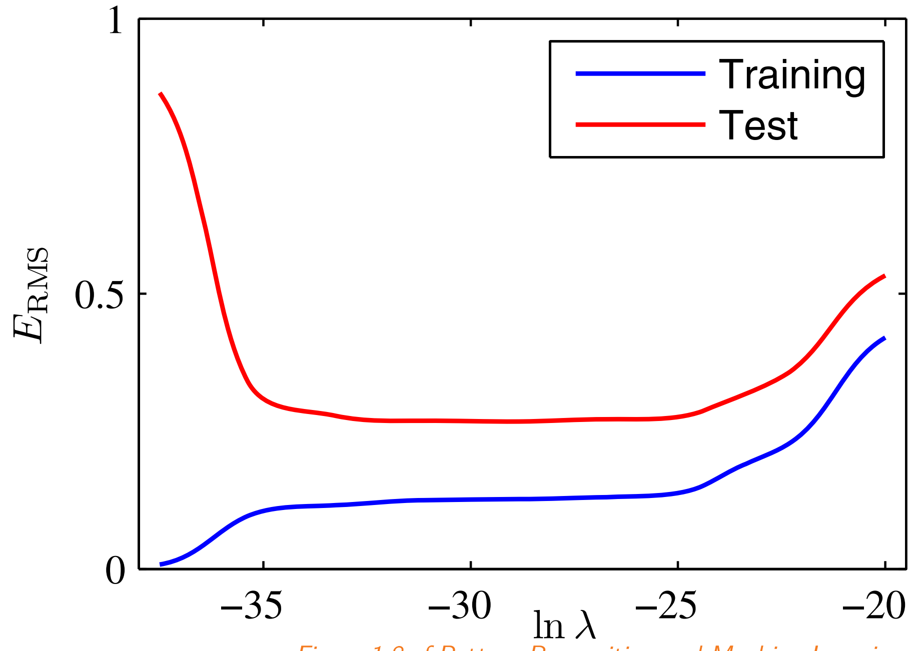


Figure 1.8 of Pattern Recognition and Machine Learning.

Linear Regression

When training a linear regression model, we minimized the sum of squares error function by computing its gradient (partial derivatives with respect to all weights) and found solution when it is equal to zero, arriving at the following equation for optimal weights:

$$\boldsymbol{X}^T \boldsymbol{X} \boldsymbol{w} = \boldsymbol{X}^T \boldsymbol{t}.$$

If $\boldsymbol{X}^T \boldsymbol{X}$ is regular, we can invert it and compute the weights as $\boldsymbol{w} = (\boldsymbol{X}^T \boldsymbol{X})^{-1} \boldsymbol{X}^T \boldsymbol{t}$.

It can be proven (see next slide) that $\operatorname{rank}(\boldsymbol{X}) = \operatorname{rank}(\boldsymbol{X}^T \boldsymbol{X})$, so matrix $\boldsymbol{X}^T \boldsymbol{X} \in \mathbb{R}^{D \times D}$ is regular if and only if $\boldsymbol{X}$ has rank D, which is equivalent to the columns of $\boldsymbol{X}$ being linearly independent.

Linear Regression Solution Always Exists

We now show that the solution of $\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{w} = \boldsymbol{X}^T\boldsymbol{t}$ always exists.

Recall that the rank-nullity theorem states that for a matrix $\boldsymbol{A} \in \mathbb{R}^{V\times W}$,

$$\operatorname{rank}(\boldsymbol{A}) + \operatorname{nullity}(\boldsymbol{A}) \overset{\text{def}}{=} \dim(\operatorname{im}(\boldsymbol{A})) + \dim(\ker(\boldsymbol{A})) = W.$$

Our goal is to show that $\operatorname{im}(\boldsymbol{X}^T\boldsymbol{X}) = \operatorname{im}(\boldsymbol{X}^T)$. Then the solution will always exist, because for any $\boldsymbol{t}$, $\boldsymbol{X}^T\boldsymbol{t} \in \operatorname{im}(\boldsymbol{X}^T\boldsymbol{X})$.

- We first show that $\ker(\boldsymbol{X}^T\boldsymbol{X}) = \ker(\boldsymbol{X})$.
  - If $\boldsymbol{X}\boldsymbol{t} = 0$, then also $\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{t} = 0$, so $\ker(\boldsymbol{X}^T\boldsymbol{X}) \supseteq \ker(\boldsymbol{X})$.
  - If $\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{t} = 0$, then also $\boldsymbol{t}^T\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{t} = 0$, therefore $(\boldsymbol{X}\boldsymbol{t})^T(\boldsymbol{X}\boldsymbol{t}) = 0$. That implies $\boldsymbol{X}\boldsymbol{t} = 0$, resulting in $\ker(\boldsymbol{X}^T\boldsymbol{X}) \subseteq \ker(\boldsymbol{X})$.
- The rank-nullity theorem therefore implies that $\operatorname{rank}(\boldsymbol{X}^T\boldsymbol{X}) = \operatorname{rank}(\boldsymbol{X}) = \operatorname{rank}(\boldsymbol{X}^T)$.
- Finally, it is easy to see that $\operatorname{im}(\boldsymbol{X}^T\boldsymbol{X}) \subseteq \operatorname{im}(\boldsymbol{X}^T)$, which combined with the equality of ranks above proves the required equation $\operatorname{im}(\boldsymbol{X}^T\boldsymbol{X}) = \operatorname{im}(\boldsymbol{X}^T)$.

SVD Solution of Linear Regression

Now consider the case that $\boldsymbol{X}^T\boldsymbol{X}$ is singular. We already know that $\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{w} = \boldsymbol{X}^T\boldsymbol{t}$ is solvable, but it does not have a unique solution (it has many solutions). Our goal in this case will be to find the $\boldsymbol{w}$ with minimum $\|\boldsymbol{w}\|$ fulfilling the equation.

We now consider *singular value decomposition (SVD)* of $\boldsymbol{X}$, writing $\boldsymbol{X} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$, where

- $\boldsymbol{U} \in \mathbb{R}^{N\times N}$ is an orthogonal matrix, i.e., $\boldsymbol{u}_i^T\boldsymbol{u}_j = [i = j] \Leftrightarrow \boldsymbol{U}^T\boldsymbol{U} = \boldsymbol{I} \Leftrightarrow \boldsymbol{U}^{-1} = \boldsymbol{U}^T$,
- $\boldsymbol{\Sigma} \in \mathbb{R}^{N\times D}$ is a diagonal matrix,
- $\boldsymbol{V} \in \mathbb{R}^{D\times D}$ is again an orthogonal matrix.

Assuming the diagonal matrix $\boldsymbol{\Sigma}$ has rank r, we can write it as

$$\boldsymbol{\Sigma} = \begin{bmatrix} \boldsymbol{\Sigma}_r & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} \end{bmatrix},$$

where $\boldsymbol{\Sigma}_r \in \mathbb{R}^{r\times r}$ is a regular diagonal matrix. Denoting $\boldsymbol{U}_r$ and $\boldsymbol{V}_r$ the matrix of first r columns of $\boldsymbol{U}$ and $\boldsymbol{V}$, respectively, we can write $\boldsymbol{X} = \boldsymbol{U}_r\boldsymbol{\Sigma}_r\boldsymbol{V}_r^T$.

SVD Solution of Linear Regression

Using the decomposition $\boldsymbol{X} = \boldsymbol{U}_r \boldsymbol{\Sigma}_r \boldsymbol{V}_r^T$, we can rewrite the goal equation as

$$\boldsymbol{V}_r \boldsymbol{\Sigma}_r^T \boldsymbol{U}_r^T \boldsymbol{U}_r \boldsymbol{\Sigma}_r \boldsymbol{V}_r^T \boldsymbol{w} = \boldsymbol{V}_r \boldsymbol{\Sigma}_r^T \boldsymbol{U}_r^T \boldsymbol{t}.$$

The transposition of an orthogonal matrix is its inverse. Therefore, our submatrix $\boldsymbol{U}_r$ fulfills $\boldsymbol{U}_r^T \boldsymbol{U}_r = \boldsymbol{I}$, because $\boldsymbol{U}_r^T \boldsymbol{U}_r$ is a top left submatrix of $\boldsymbol{U}^T \boldsymbol{U}$. Analogously, $\boldsymbol{V}_r^T \boldsymbol{V}_r = \boldsymbol{I}$.

We therefore simplify the goal equation to

$$\boldsymbol{\Sigma}_r \boldsymbol{\Sigma}_r \boldsymbol{V}_r^T \boldsymbol{w} = \boldsymbol{\Sigma}_r \boldsymbol{U}_r^T \boldsymbol{t}$$

Because the diagonal matrix $\boldsymbol{\Sigma}_r$ is regular, we can divide by it and obtain

$$\boldsymbol{V}_r^T \boldsymbol{w} = \boldsymbol{\Sigma}_r^{-1} \boldsymbol{U}_r^T \boldsymbol{t}.$$

SVD Solution of Linear Regression

We have $\boldsymbol{V}_r^T\boldsymbol{w} = \boldsymbol{\Sigma}_r^{-1}\boldsymbol{U}_r^T\boldsymbol{t}$. If the original matrix $\boldsymbol{X}^T\boldsymbol{X}$ was regular, then $r = D$ and $\boldsymbol{V}_r$ is a square regular orthogonal matrix, in which case

$$\boldsymbol{w} = \boldsymbol{V}_r\boldsymbol{\Sigma}_r^{-1}\boldsymbol{U}_r^T\boldsymbol{t}.$$

If we denote $\boldsymbol{\Sigma}^+ \in \mathbb{R}^{D\times N}$ the diagonal matrix with $\Sigma_{i,i}^{-1}$ on diagonal, we can rewrite $\boldsymbol{w}$ to

$$\boldsymbol{w} = \boldsymbol{V}\boldsymbol{\Sigma}^+\boldsymbol{U}^T\boldsymbol{t}.$$

Now if $r < D$, $\boldsymbol{V}_r^T\boldsymbol{w} = \boldsymbol{y}$ is undetermined and has infinitely many solutions. To find the one with smallest norm $\|\boldsymbol{w}\|$, consider the full product $\boldsymbol{V}^T\boldsymbol{w}$. Because $\boldsymbol{V}$ is orthogonal, $\|\boldsymbol{V}^T\boldsymbol{w}\| = \|\boldsymbol{w}\|$, and it is sufficient to find $\boldsymbol{w}$ with smallest $\|\boldsymbol{V}^T\boldsymbol{w}\|$. We know that the first r elements of $\|\boldsymbol{V}^T\boldsymbol{w}\|$ are fixed by the above equation – the smallest $\|\boldsymbol{V}^T\boldsymbol{w}\|$ can be therefore obtained by setting the last $D-r$ elements to zero. Finally, we note that $\boldsymbol{\Sigma}^+\boldsymbol{U}^T\boldsymbol{t}$ is exactly $\boldsymbol{\Sigma}_r^{-1}\boldsymbol{U}_r^T\boldsymbol{t}$ padded with $D-r$ zeros, obtaining the same solution $\boldsymbol{w} = \boldsymbol{V}\boldsymbol{\Sigma}^+\boldsymbol{U}^T\boldsymbol{t}$.

SVD Solution of Linear Regression and Pseudoinverses

The solution to a linear regression with sum of squares error function is tightly connected to matrix pseudoinverses. If a matrix $\boldsymbol{X}$ is singular or rectangular, it does not have an exact inverse, and $\boldsymbol{X}\boldsymbol{w} = \boldsymbol{b}$ does not have an exact solution.

However, we can consider the so-called *Moore-Penrose pseudoinverse*

$$\boldsymbol{X}^+ \stackrel{\text{def}}{=} \boldsymbol{V}\boldsymbol{\Sigma}^+\boldsymbol{U}^T$$

to be the closest approximation to an inverse, in the sense that we can find the best solution (with smallest MSE) to the equation $\boldsymbol{X}\boldsymbol{w} = \boldsymbol{b}$ by setting $\boldsymbol{w} = \boldsymbol{X}^+\boldsymbol{b}$.

Alternatively, we can define the pseudoinverse of a matrix $\boldsymbol{X}$ as

$$\boldsymbol{X}^+ = \underset{\boldsymbol{Y}\in\mathbb{R}^{D\times N}}{\arg\min} \left\|\boldsymbol{X}\boldsymbol{Y} - \boldsymbol{I}_N\right\|_F = \underset{\boldsymbol{Y}\in\mathbb{R}^{D\times N}}{\arg\min} \left\|\boldsymbol{Y}\boldsymbol{X} - \boldsymbol{I}_D\right\|_F$$

which can be verified to be the same as our SVD formula.

Random Variables

A random variable x is a result of a random process. It can be discrete or continuous.

Probability Distribution

A probability distribution describes how likely are individual values a random variable can take.

The notation $\mathrm{x} \sim P$ stands for a random variable x having a distribution P.

For discrete variables, the probability that x takes a value x is denoted as $P(x)$ or explicitly as $P(\mathrm{x} = x)$. All probabilities are non-negative and sum of probabilities of all possible values of x is $\sum_x P(\mathrm{x} = x) = 1$.

For continuous variables, the probability that the value of x lies in the interval $[a, b]$ is given by $\int_a^b p(x)\,\mathrm{d}x$, where $p(x)$ is the *probability density function*, which is always non-negative and integrates to 1 over the range of all values of x.

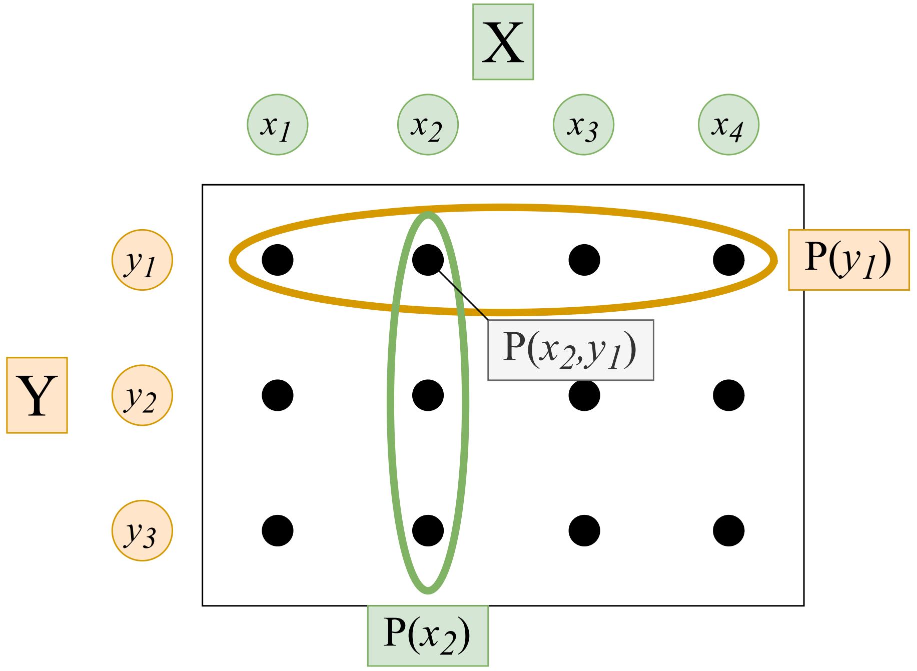
Joint, Conditional, Marginal Probability

For two random variables, **joint probability distribution** is a distribution of all possible pairs of outputs (and analogously for more than two):

$$P(\mathrm{x} = x_2, \mathrm{y} = y_1).$$

Marginal distribution is a distribution of one (or a subset) of the random variables and can be obtained by summing over the other variable(s):

$$P(\mathrm{x} = x_2) = \sum\nolimits_y P(\mathrm{x} = x_2, \mathrm{y} = y).$$



Conditional distribution is a distribution of one (or a subset) of the random variables, given that another event has already occurred:

$$P(\mathrm{x} = x_2 | \mathrm{y} = y_1) = P(\mathrm{x} = x_2, \mathrm{y} = y_1)/P(\mathrm{y} = y_1).$$

If $P(\mathrm{x}, \mathrm{y}) = P(\mathrm{x}) \cdot P(\mathrm{y})$ or $P(\mathrm{x}|\mathrm{y}) = P(\mathrm{x})$, random variables x and y are **independent**.

Random Variables

Expectation

The expectation of a function $f(x)$ with respect to discrete probability distribution $P(x)$ is defined as:

$$\mathbb{E}_{\mathrm{x} \sim P}[f(x)] \stackrel{\text{def}}{=} \sum_x P(x) f(x).$$

For continuous variables it is computed as:

$$\mathbb{E}_{\mathrm{x} \sim p}[f(x)] \stackrel{\text{def}}{=} \int_x p(x) f(x) \,\mathrm{d}x.$$

If the random variable is obvious from context, we can write only $\mathbb{E}_P[x]$ or even $\mathbb{E}[x]$.

Expectation is linear, i.e.,

$$\mathbb{E}_{\mathrm{x}}[\alpha f(x) + \beta g(x)] = \alpha \mathbb{E}_{\mathrm{x}}[f(x)] + \beta \mathbb{E}_{\mathrm{x}}[g(x)].$$

Random Variables

Variance

Variance measures how much the values of a random variable differ from its mean $\mu = \mathbb{E}[x]$.

$$\mathrm{Var}(x) \overset{\text{def}}{=} \mathbb{E}\left[\big(x - \mathbb{E}[x]\big)^2\right], \text{ or more generally,}$$
$$\mathrm{Var}(f(x)) \overset{\text{def}}{=} \mathbb{E}\left[\big(f(x) - \mathbb{E}[f(x)]\big)^2\right].$$

It is easy to see that

$$\mathrm{Var}(x) = \mathbb{E}\left[x^2 - 2x\mathbb{E}[x] + \big(\mathbb{E}[x]\big)^2\right] = \mathbb{E}\left[x^2\right] - \big(\mathbb{E}[x]\big)^2,$$

because $\mathbb{E}\big[2x\mathbb{E}[x]\big] = 2(\mathbb{E}[x])^2$.

Variance is connected to $E[x^2]$, the **second moment** of a random variable – it is in fact a **centered** second moment.

Estimators and Bias

An **estimator** is a rule for computing an estimate of a given value, often an expectation of some random value(s).

For example, we might estimate *mean* of random variable by sampling a value according to its probability distribution.

Bias of an estimator is the difference of the expected value of the estimator and the true value being estimated:

$$\text{bias} = \mathbb{E}[\text{estimate}] - \text{true estimated value}.$$

If the bias is zero, we call the estimator **unbiased**, otherwise we call it **biased**.

As an example, consider estimating $\mathbb{E}_P[f(x)]$ by generating a single sample x from P and returning $f(x)$. Such an estimate is unbiased, because $\mathbb{E}[\text{estimate}] = \mathbb{E}_P[f(x)]$, which is exactly the true estimated value.

Estimators and Bias

If we have a sequence of estimates, it also might happen that the bias converges to zero. Consider the well known sample estimate of variance. Given $\mathbf{x}_1, \ldots, \mathbf{x}_n$ independent and identically distributed random variables, we might estimate mean and variance as

$$\hat\mu = \frac{1}{n}\sum\nolimits_i x_i, \quad \hat\sigma^2 = \frac{1}{n}\sum\nolimits_i (x_i - \hat\mu)^2.$$

Such estimate is biased, because $\mathbb{E}[\hat\sigma^2] = (1 - \frac{1}{n})\sigma^2$, but the bias converges to zero with increasing n.

Also, an unbiased estimator does not necessarily have small variance – in some cases it can have large variance, so a biased estimator with smaller variance might be preferred.

Gradient Descent

Sometimes it is more practical to search for the best model weights in an iterative/incremental/sequential fashion. Either because there is too much data, or the direct optimization is not feasible.

Assuming we are minimizing an error function

$$\arg\min_{\boldsymbol w} E(\boldsymbol w),$$

we may use *gradient descent*:

$$\boldsymbol w \leftarrow \boldsymbol w - \alpha \nabla_{\boldsymbol w} E(\boldsymbol w)$$

The constant α is called a **learning rate** and specifies the "length" of a step we perform in every iteration of the gradient descent.

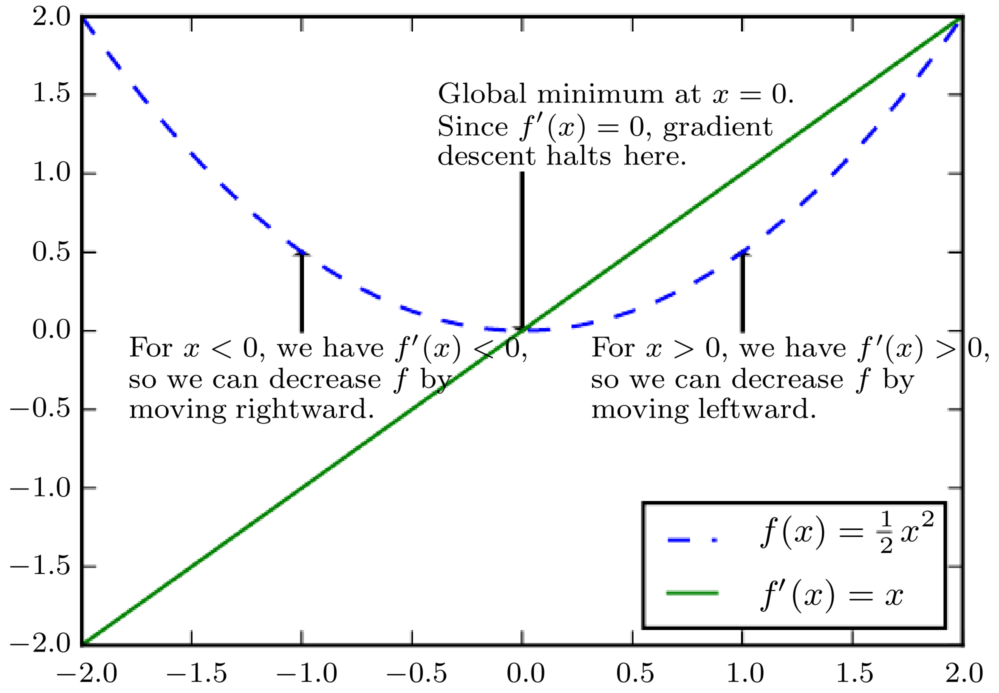


Figure 4.1, page 83 of Deep Learning Book, http://deeplearningbook.org

Gradient Descent Variants

Consider an error function computed as an expectation over the dataset:

$$E(\boldsymbol{w}) = \mathbb{E}_{(\boldsymbol{x},t)\sim\hat{p}_{\text{data}}} L\big(y(\boldsymbol{x};\boldsymbol{w}), t\big), \text{ so that } \nabla_{\boldsymbol{w}} E(\boldsymbol{w}) = \mathbb{E}_{(\boldsymbol{x},t)\sim\hat{p}_{\text{data}}} \nabla_{\boldsymbol{w}} L\big(y(\boldsymbol{x};\boldsymbol{w}), t\big).$$

- **(Standard/Batch) Gradient Descent**: We use all training data to compute $\nabla_{\boldsymbol{w}} E(\boldsymbol{w})$.
- **Stochastic (or Online) Gradient Descent**: We estimate $\nabla_{\boldsymbol{w}} E(\boldsymbol{w})$ using a single random example from the training data. Such an estimate is unbiased, but very noisy.

$$\nabla_{\boldsymbol{w}} E(\boldsymbol{w}) \approx \nabla_{\boldsymbol{w}} L\big(y(\boldsymbol{x};\boldsymbol{w}), t\big) \text{ for randomly chosen } (\boldsymbol{x}, t) \text{ from } \hat{p}_{\text{data}}.$$

- **Minibatch SGD**: The minibatch SGD is a trade-off between gradient descent and SGD – the expectation in $\nabla_{\boldsymbol{w}} E(\boldsymbol{w})$ is estimated using B random independent examples from the training data.

$$\nabla_{\boldsymbol{w}} E(\boldsymbol{w}) \approx \frac{1}{B} \sum_{i=1}^{B} \nabla_{\boldsymbol{w}} L\big(y(\boldsymbol{x}_i;\boldsymbol{w}), t_i\big) \text{ for randomly chosen } (\boldsymbol{x}_i, t_i) \text{ from } \hat{p}_{\text{data}}.$$

Gradient Descent Convergence

Assume that we perform a stochastic gradient descent, using a sequence of learning rates α_i, and using a noisy estimate $J(\boldsymbol w)$ of the real gradient $\nabla_{\boldsymbol w} E(\boldsymbol w)$:

$$\boldsymbol w_{i+1} \leftarrow \boldsymbol w_i - \alpha_i J(\boldsymbol w_i).$$

It can be proven (under some reasonable conditions; see Robbins and Monro algorithm, 1951) that if the loss function L is convex and continuous, then SGD converges to the unique optimum almost surely if the sequence of learning rates α_i fulfills the following conditions:

$$\forall i : \alpha_i > 0,\ \ \sum_i \alpha_i = \infty,\ \ \sum_i \alpha_i^2 < \infty.$$

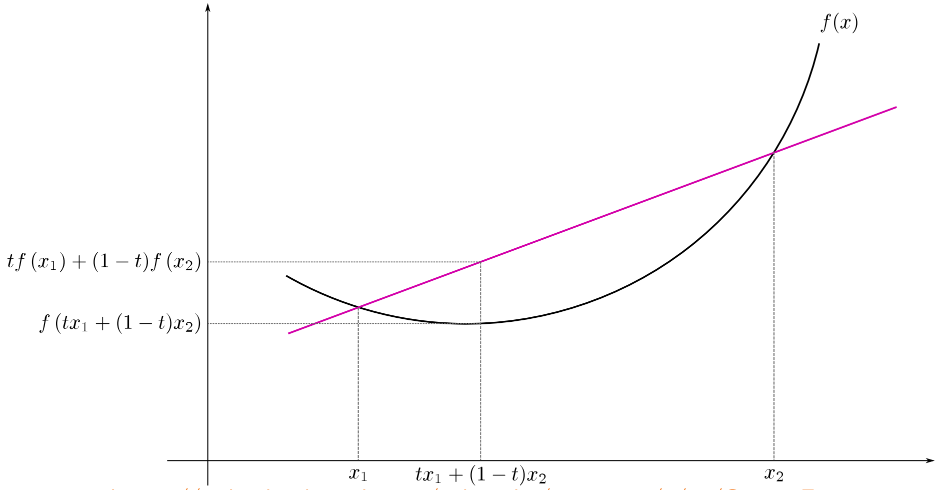
Note that the third condition implies that $\alpha_i \rightarrow 0$.

For non-convex loss functions, we can get guarantees of converging to a *local* optimum only. However, note that finding a global minimum of an arbitrary function is *at least NP-hard*.

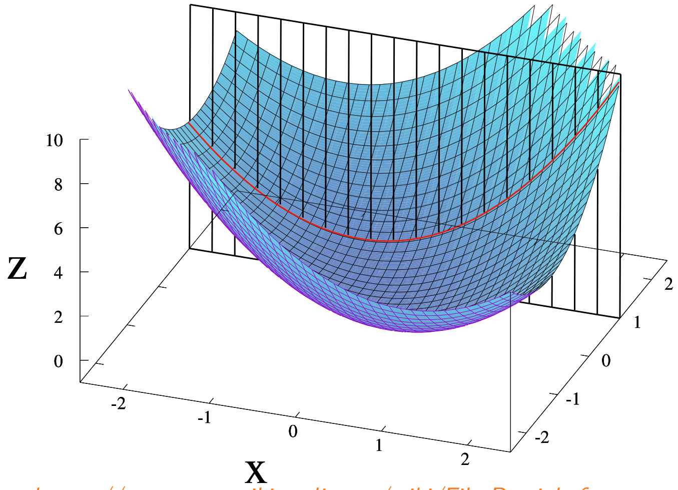
Gradient Descent Convergence

Convex functions mentioned on a previous slide are such that for x_1, x_2 and real $0 \leq t \leq 1$,

$$f(tx_1 + (1-t)x_2) \leq tf(x_1) + (1-t)f(x_2).$$



https://upload.wikimedia.org/wikipedia/commons/c/c7/ConvexFunction.svg



https://commons.wikimedia.org/wiki/File:Partial_func_eg.svg

A twice-differentiable function is convex iff its second derivative is always non-negative.

A local minimum of a convex function is always the unique global minimum.

Well-known examples of convex functions are x^2, e^x and $-\log x$.

Solving Linear Regression using SGD

To apply SGD on linear regression, we usually minimize one half of mean squared error:

$$E(\boldsymbol{w}) = \mathbb{E}_{(\boldsymbol{x},t)\sim\hat{p}_\text{data}}\left[\tfrac{1}{2}(y(\boldsymbol{x};\boldsymbol{w}) - t)^2\right] = \mathbb{E}_{(\boldsymbol{x},t)\sim\hat{p}_\text{data}}\left[\tfrac{1}{2}(\boldsymbol{x}^T\boldsymbol{w} - t)^2\right].$$

If we also include L_2 regularization, we get

$$E(\boldsymbol{w}) = \mathbb{E}_{(\boldsymbol{x},t)\sim\hat{p}_\text{data}}\left[\tfrac{1}{2}(\boldsymbol{x}^T\boldsymbol{w} - t)^2\right] + \tfrac{\lambda}{2}\|\boldsymbol{w}\|^2.$$

We then estimate the expectation by a minibatch of examples with indices $\boldsymbol{b}$ as

$$\sum_{i\in\boldsymbol{b}} \frac{1}{|\boldsymbol{b}|}\Big(\tfrac{1}{2}(\boldsymbol{x}_i^T\boldsymbol{w} - t_i)^2\Big) + \tfrac{\lambda}{2}\|\boldsymbol{w}\|^2,$$

which gives us an estimate of a gradient

$$\nabla_{\boldsymbol{w}} E(\boldsymbol{w}) \approx \sum_{i\in\boldsymbol{b}} \frac{1}{|\boldsymbol{b}|}\Big((\boldsymbol{x}_i^T\boldsymbol{w} - t_i)\boldsymbol{x}_i\Big) + \lambda\boldsymbol{w}.$$

Solving Linear Regression using SGD

The computed gradient allows us to formulate the following algorithm for solving linear regression with minibatch SGD.

Input: Dataset ($\boldsymbol{X} \in \mathbb{R}^{N\times D}$, $\boldsymbol{t} \in \mathbb{R}^N$), learning rate $\alpha \in \mathbb{R}^+$, L_2 strength $\lambda \in \mathbb{R}$.
Output: Weights $\boldsymbol{w} \in \mathbb{R}^D$ which hopefully minimize regularized MSE of linear regression.

- $\boldsymbol{w} \leftarrow 0$
- repeat until convergence (or until our patience runs out):
  - sample a minibatch of examples with indices $\boldsymbol{b}$
    - either uniformly randomly,
    - or we may want to process all training instances before repeating them, which can be implemented by generating a random permutation and then splitting it to minibatch-sizes chunks
      - the most common option; one pass through the data is called an **epoch**
  - $\boldsymbol{w} \leftarrow \boldsymbol{w} - \alpha \sum_{i\in\boldsymbol{b}} \frac{1}{|\boldsymbol{b}|}\big((\boldsymbol{x}_i^T \boldsymbol{w} - t_i)\boldsymbol{x}_i\big) - \alpha\lambda\boldsymbol{w}$

Features

Recall that the *input* instance values are usually the raw observations and are given. However, we might extend them suitably before running a machine learning algorithm, especially if the algorithm is linear or otherwise limited and cannot represent arbitrary function. Such instance representations is called *features*.

We already saw this in the example from the previous lecture, where even if our training examples were x and t, we performed the linear regression using features $(x^0, x^1, \ldots, x^M)$:

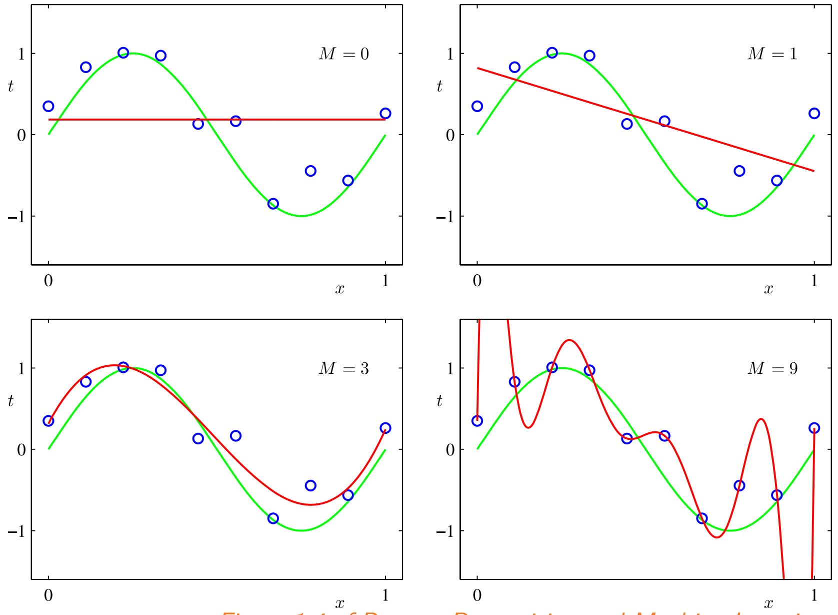


Figure 1.4 of Pattern Recognition and Machine Learning.

Features

Generally, it would be best if we have machine learning algorithms processing only the raw inputs. However, many algorithms are capable of representing only a limited set of functions (for example linear ones), and in that case, *feature engineering* plays a major part in the final model performance. Feature engineering is a process of constructing features from raw inputs.

Commonly used features are:

- **polynomial features** of degree p: Given features $(x_1, x_2, \ldots, x_D)$, we might consider *all* products of p input values. Therefore, polynomial features of degree 2 would consist of $x_i^2\ \forall i$ and of $x_i x_j\ \forall i \neq j$.
- **categorical one-hot features**: Assume for example that a day in a week is represented on the input as an integer value of 1 to 7, or a breed of a dog is expressed as an integer value of 0 to 366. Using these integral values as input to linear regression makes little sense – instead it might be better to learn weights for individual days in a week or for individual dog breeds. We might therefore represent input classes by binary indicators for every class, giving rise to **one-hot** representation, where input integral value $1 \leq v \leq L$ is represented as L binary values, which are all zero except for the v-th one, which is one.